Real-Time Migration Properties of rTiMo Verified in Uppaal

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Abstract. This paper extends the TiMo family by introducing a real-time version named rTiMo. The rTiMo processes are able to move between different locations of a distributed environment, and communicate locally with other processes. Real-time constraints are used to control migration and communication in a real-time distributed system. In order to verify several properties of complex mobile systems described in rTiMo, we establish a relationship between rTiMo networks and a class of timed safety automata. The relationship allows the verification of temporal properties of real-time migrating processes using Uppaal capabilities. In particular, we check whether certain configurations are reached, and that certain timing constraints hold for an entire complex evolution.

1 Introduction

A rather simple and expressive formalism called TiMo was previously introduced in [8] in order to describe complex distributed systems in which processes are able to migrate within an environment defined by a number of explicit locations. Processes are active entities that can move from location to location to meet and communicate with other processes rather than using the client/server method (various types of communication are presented in [4]). Each process has its own agenda and hence initiates and controls its interactions according to its needs and goals. Timing constraints are used to coordinate interactions in time and space by using migration and communication [9]. Timing constraints for migration allow one to specify a temporal interval after which a mobile process must move to another location. Two processes may communicate if they are present at the same location. Inspired by TiMo, a flexible software platform supporting the specification of agents and allowing timed migration in a distributed environment is presented in [7]. We have enriched this basic formalism with access permissions by using a type system [10]. More information on the TiMo family is available at `iit.iit.tuiasi.ro/~fml/TiMo`.

This paper is devoted to a real-time extension of TiMo named rTiMo, a calculus in which a global clock is used for the dynamic evolution of the whole system. In rTiMo, the discrete transitions caused by performing actions with timeouts are alternated with continuous transitions. Although the syntax of rTiMo is close to that of TiMo [10], their semantics are different. The number of semantic rules in rTiMo is higher than in TiMo . Other differences between rTiMo and TiMo are:

R.M. Hierons, M.G. Merayo, and M. Bravetti (Eds.): SEFM 2013, LNCS 8137, pp. 31–45, 2013.

- *action deadline* in rTiMo is a real positive number, while in TiMo it is a positive natural number;
- *clock* in rTiMo is a single global clock, while in TiMo there is a local clock for each location;
- *time step* in rTiMo can have any length, while in TiMo it has length 1 (at each location);
- *passage of time* in rTiMo is performed by delay rules, in contrast with TiMo where in each location l there is a local function ϕ_l that is used to decrement all timers by 1 at location l;
- *evolution step* in rTiMo is a sequence of individual actions followed by the passing of time, in contrast with TiMo where an evolution step is a sequence of individual actions happening at the same location l, followed by the passing of time and elimination of all special symbols Ⓢ at location l (Ⓢ is a purely technical notation used in the formalisation of the structural operational semantics of TiMo; intuitively, ⓈP specifies a process P that is temporarily stalled and so cannot execute any action).

The semantics of rTiMo is provided by multiset labelled transitions in which multisets indicate the actions executed in parallel. In order to illustrate the coordination in time and space of a migrating process in rTiMo, we adapt the *TravelShop* example used in [9] where the clients buy tickets to predefined destinations from some travel agents. Since time is an important issue, it was studied in various papers [11,13]. Within rTiMo we investigate the possibility of verifying certain interesting real-time properties such as safety properties (a specified error cannot occur) and bounded liveness properties (configuration reachability within a certain amount of time). The development of effective techniques and tools is required by the automated analysis and verification of complex distributed systems. We establish a formal relationship between rTiMo and timed safety automata [15], allowing the use of the model checking capabilities of the software tool Uppaal [17] to verify several temporal properties of distributed networks with migrating and communicating processes described in rTiMo.

2 Syntax and Semantics of rTiMo

The syntax of rTiMo is given in Table 1, where we assume:

- a set *Loc* of locations, a set *Chan* of communication channels, and a set *Id* of process identifiers (each $id \in Id$ has its arity m_{id});
- for each $id \in Id$ there is a unique process definition $id(u_1, \ldots, u_{m_{id}}) \stackrel{def}{=} P_{id}$, where the distinct variables u_i are parameters;
- $a \in Chan$ is a communication channel; l is a location or a location variable;
- $t \in \mathbb{R}_+$ is a *timeout* (deadline) of an action; u is a tuple of variables;
- v is a tuple of expressions built from values, variables and allowed operations.

Timing constraints applied to migrating processes allow one to specify how many time units are required by a process to move from one location to another.

A timer in rTiMo is denoted by $^{\Delta 3}$. When it is associated with a migration process $go^{\Delta 3} shop$ then P, it indicates that process P moves to location *shop* after 3 time units. A timer $^{\Delta 5}$ associated with an output process $a^{\Delta 5}!\langle z\rangle$ then P else Q makes the channel a available for communication, namely it can send z for a period of 5 time units. It is also possible to restrict the waiting time for an input process $a^{\Delta 4}?(x)$ then P else Q along a channel a; if the communication does not happen before the deadline 4, the waiting process gives up and it switches to the alternative process Q.

Table 1. rTiMo Syntax

Processes	P, Q	$::= a^{\Delta t}!\langle v\rangle$ then P else Q ∣	(output)
		$a^{\Delta t}?(u)$ then P else Q ∣	(input)
		$go^{\Delta t} l$ then P ∣	(move)
		0 ∣	(termination)
		$id(v)$	(recursion)
		$P \mid Q$ ∣	(parallel)
Located processes	L	$::= l[[P]]$	
Networks	N	$::= L$ ∣ $L \mid N$	

The only variable binding constructor is $a^{\Delta t}?(u)$ then P else Q which binds the variable u within P (but *not* within Q). We use $fv(P)$ to denote the free variables of a process P (and similarly for networks); for a process definition, we assume that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$, where u_i are the process parameters. Processes are defined up-to an alpha-conversion, and $\{v/u, \ldots\}P$ denotes P in which all free occurrences of a variable u are replaced by v, possible after alpha-converting P in order to avoid clashes.

Mobility is provided by a process $go^{\Delta t} l$ then P that describes the migration from the current location to the location indicated by l within t time units. Since l can be a variable, and so its value is assigned dynamically through communication with other processes, this form of migration supports a flexible scheme for the movement of processes from one location to another. Thus, the behaviour can adapt to various changes of the distributed environment. Processes are further constructed from the (terminated) process 0, and parallel composition $P \mid Q$. A located process $l[[P]]$ specifies a process P running at location l, and a network is built from its components $N \mid N'$. A network N is well-formed if there are no free variables in N.

The first component of the operational semantics of rTiMo is the structural equivalence $\equiv$ on networks; it is the smallest congruence such that the equalities in Table 2 hold.

Table 2. rTiMo Structural Congruence

(NNULL)	$N \mid 0 \equiv N$
(NCOMM)	$N \mid N' \equiv N' \mid N$
(NASSOC)	$(N \mid N') \mid N'' \equiv N \mid (N' \mid N'')$
(NSPLIT)	$l[[P \mid Q]] \equiv l[[P]] \mid l[[Q]]$

The role of $\equiv$ is to rearrange a network in order to apply the rules of the operational semantics given in Table 3. Using the equalities of Table 2, a given network N can always be transformed into a finite parallel composition of located processes of the form $l_1[[P_1]] \mid \ldots \mid l_n[[P_n]]$ such that no process P_i has the parallel composition operator at its topmost level. Each located process $l_i[[P_i]]$ is called a component of N, and the whole expression $l_1[[P_1]] \mid \ldots \mid l_n[[P_n]]$ is called a *component decomposition* of the network N.

The semantics of rTiMo is presented in Table 3. The multiset labelled transitions of form $N \xrightarrow{\Lambda} N'$ use a multiset Λ to indicate the actions executed in parallel in one step. When the multiset Λ contains only one action λ, in order to simplify the syntax, we write $N \xrightarrow{\lambda} N'$. The transitions of form $N \overset{t}{\rightsquigarrow} N'$ represent a time step of length t.

In rule (Move0), the process $go^{\Delta t} l$ then P migrates from location l to l' and evolves as process P. In rule (Com), a process $a^{\Delta t}!\langle v\rangle$ then P else Q, from location l, succeeds in sending a tuple of values v over channel a to process $a^{\Delta t}?(u)$ then P' else Q' from location l. Both processes continue to execute at location l, the first one as P and the second one as $\{v/u\}P'$. If a communication action has a timer equal to 0, then by using the rule (Put0) for output action or the rule (Get0) for input action, the process $a^{\Delta 0} * $ then P else Q, for $* \in \{!\langle v\rangle, ?(x)\}$ continues as the alternative process Q. Rule (Call) simulates the evolution of a recursion process. The rules (Equiv) and (DEquiv) are used to rearrange a network in order to apply a rule. Rule (Par) is used to compose larger networks from smaller ones by putting them in parallel and considering the union of multisets of actions.

The rules devoted to the passing of time are starting with D. In rule (DPar), $N_1 \mid N_2 \overset{\lambda}{\not\rightarrow}$ means that no action λ (i.e, an action labelled by $l' \triangleright l$, $\{v/u\}@l$, $id@l$, $go^{\Delta 0}@l$, $a?^{\Delta 0}@l$ or $a!^{\Delta 0}@l$) can be applied to the network $N_1 \mid N_2$ (obtained using (Par) rules). We use negative premises: the passing to a new step is performed based on the absence of actions. According to [12], our semantics allows the use of negative premises without leading to an inconsistent set of rules.

A complete computational step is captured by a derivation of the form:

$$N \xrightarrow{\Lambda} N_1 \overset{t}{\rightsquigarrow} N'.$$

This means that a derivation is a sequence of individual actions followed by a time step. We say that N' is directly reachable from N. If there is no applicable action, we write $N \overset{t}{\rightsquigarrow} N'$ to indicate time progress.

The first item of the following proposition states that the passage of time does not introduce any nondeterminism into the execution of a process. Also, if a process is able to evolve to a certain time t, then it must evolve through every time moment before t; this means that the process evolves continuously.

Proposition 1. *For any networks N, N' and N'', the following sentences hold:*

1. $N \overset{0}{\rightsquigarrow} N$;
2. *If* $N \overset{t}{\rightsquigarrow} N'$ *and* $N \overset{t}{\rightsquigarrow} N''$, *then* $N' \equiv N''$;
3. $N \overset{(t+t')}{\rightsquigarrow} N'$ *if and only if there is a* N'' *such that* $N \overset{t}{\rightsquigarrow} N''$ *and* $N'' \overset{t'}{\rightsquigarrow} N'$.

Table 3. rTiMo Operational Semantics

(Stop) $l[[0]] \not\xrightarrow{\lambda}$ (DStop) $l[[0]] \overset{t}{\rightsquigarrow} l[[0]]$

(DMove) $$\frac{t \geq t' \geq 0}{l[[go^{\Delta t} l' \text{ then } P]] \overset{t'}{\rightsquigarrow} l[[go^{\Delta t - t'} l' \text{ then } P]]}$$

(Move0) $$l[[go^{\Delta 0} l' \text{ then } P]] \xrightarrow{l \triangleright l'} l'[[P]]$$

(Com) $$l[[a^{\Delta t}!\langle v\rangle \text{ then } P \text{ else } Q \mid a^{\Delta t'}?(u) \text{ then } P' \text{ else } Q']] \xrightarrow{\{v/u\}@l} l[[P \mid \{v/u\}P']]$$

(DPut) $$\frac{t \geq t' \geq 0}{l[[a^{\Delta t}!\langle v\rangle \text{ then } P \text{ else } Q]] \overset{t'}{\rightsquigarrow} l[[a^{\Delta t - t'}!\langle v\rangle \text{ then } P \text{ else } Q]]}$$

(Put0) $$l[[a^{\Delta 0}!\langle v\rangle \text{ then } P \text{ else } Q]] \xrightarrow{a!^{\Delta 0}@l} l[[Q]]$$

(DGet) $$\frac{t \geq t' \geq 0}{l[[a^{\Delta t}?(u) \text{ then } P \text{ else } Q]] \overset{t'}{\rightsquigarrow} l[[a^{\Delta t - t'}?(u) \text{ then } P \text{ else } Q]]}$$

(Get0) $$l[[a^{\Delta 0}?(u) \text{ then } P \text{ else } Q]] \xrightarrow{a?^{\Delta 0}@l} l[[Q]]$$

(DCall) $$\frac{l[[P_{id}\{v/x\}]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]}{l[[id(v)]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]} \text{ where } id(v) \overset{def}{=} P_{id}$$

(Call) $$\frac{l[[P_{id}\{v/x\}]] \xrightarrow{\Lambda} l[[P'_{id}]]}{l[[id(v)]] \xrightarrow{\Lambda} l[[P'_{id}]]} \text{ where } id(v) \overset{def}{=} P_{id}$$

(DPar) $$\frac{N_1 \overset{t}{\rightsquigarrow} N'_1 \qquad N_2 \overset{t}{\rightsquigarrow} N'_2 \qquad N_1 \mid N_2 \not\xrightarrow{\lambda}}{N_1 \mid N_2 \overset{t}{\rightsquigarrow} N'_1 \mid N'_2}$$

(Par) $$\frac{N_1 \xrightarrow{\Lambda_1} N'_1 \quad N_2 \xrightarrow{\Lambda_2} N'_2}{N_1 \mid N_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} N'_1 \mid N'_2}$$

(DEquiv) $$\frac{N \equiv N' \qquad N' \overset{t}{\rightsquigarrow} N'' \qquad N'' \equiv N'''}{N \overset{t}{\rightsquigarrow} N'''}$$

(Equiv) $$\frac{N \equiv N' \qquad N' \xrightarrow{\Lambda} N'' \qquad N'' \equiv N'''}{N \xrightarrow{\Lambda} N'''}$$

Example 1. We adapt the *TravelShop* example of [9] in which a client attempts to get a ticket to a predefined destination in a short time and at a good price. The scenario involves five locations and six processes.

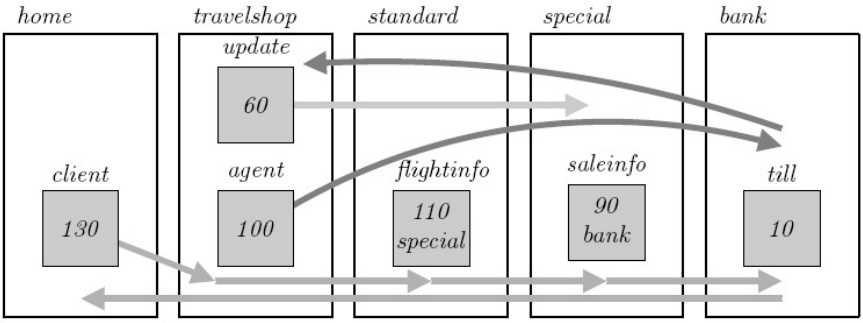


Fig. 1. The initial network indicates the migration paths of the processes [9]

The role of each process represented in Figure 1 is as follows:

- *client* is a process that initially resides in the *home* location, has an amount of 130 cash, and intends to pay for a flight after comparing two offers (standard and special) provided by the travel shop. After entering the *travelshop* location, the *client* receives the location of the standard offer where it should move to obtain this standard offer, and also the location where a special offer can be obtained. Then, it moves to the *special* location to receive the special offer. Finally, the *client* moves to the bank, pays for the special (cheaper) offer, and returns to the *home* location.
- *agent* is a process that initially resides in the *travelshop* location, has an amount of 100 cash, and informs the *client* where to look for the standard offer. It then moves to the *bank* in order to collect the money from the *till*. After that, the *agent* returns to the *travelshop*.
- *flightinfo* communicates the *standard* offer (110 cash) to clients as well as the location (*special*) of the *special* offer.
- *saleinfo* communicates the *special* offer (90 cash) to clients together with the location (*bank*) of the bank. It can also accept an update of the special offer coming from the *travelshop* location.
- *update* migrates from the *travelshop* to the *special* location in order to update the *special* offer to the amount of 60 cash.
- *till* resides at the *bank* location, has an initial amount of 10 cash, and can either receive e-money paid in by the clients, or transfer the accumulated e-money to the *agent*.

In what follows we use some shorthand notations:
$a!\langle v\rangle \to P$ stands for $a^{\Delta\infty}\langle v\rangle$ then P else 0;
$a?(u) \to P$ stands for $a^{\Delta\infty}(u)$ then P else 0.

The rTiMo syntax of these processes is as follows:

$$client(init) = \mathsf{go}^{\Delta 5} travelshop \rightarrow flight?(standardoffer) \rightarrow \mathsf{go}^{\Delta 4} standardoffer \rightarrow finfo2?(p1, specialoffer) \rightarrow \mathsf{go}^{\Delta 3} specialoffer \rightarrow sinfo2?(p2, paying) \rightarrow \mathsf{go}^{\Delta 6} paying \rightarrow payc!\langle min\{p1, p2\}\rangle \rightarrow \mathsf{go}^{\Delta 4} home \rightarrow client(init - min\{p1, p2\})$$

$$update(saleprice) = \mathsf{go}^{\Delta 0} special \rightarrow info1!\langle saleprice\rangle$$

$$agent(balance) = flight!\langle standard\rangle \rightarrow \mathsf{go}^{\Delta 10} bank \rightarrow paya?(profit) \rightarrow \mathsf{go}^{\Delta 12} travelshop \rightarrow agent(balance + profit)$$

$$flightinfo(price, next) = finfo2!\langle price, next\rangle \rightarrow flightinfo(price, next)$$

$$saleinfo(price, next) = info1^{\Delta 10}?(newprice) \text{ then } saleinfo(newprice, next) \text{ else } sinfo2!\langle price, next\rangle \rightarrow saleinfo(price, next)$$

$$till(cash) = payc^{\Delta 1}?(newpayment) \text{ then } till(cash + newpayment) \text{ else } paya!\langle cash\rangle! \text{ then } till(0) \text{ else } till(cash))$$

A possible final network after 22 units of time is represented in Figure 2.

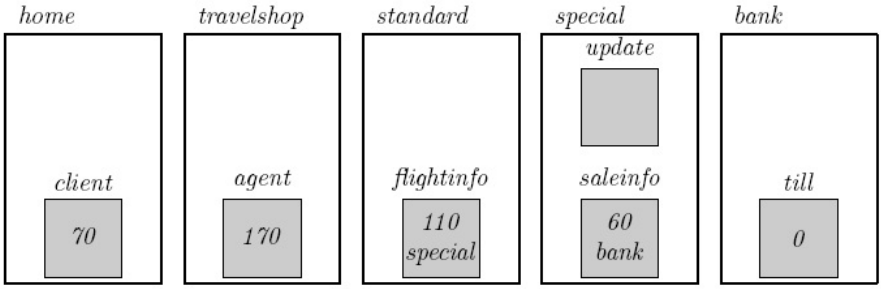


Fig. 2. A possible final network [9]

3 Timed Safety Automata

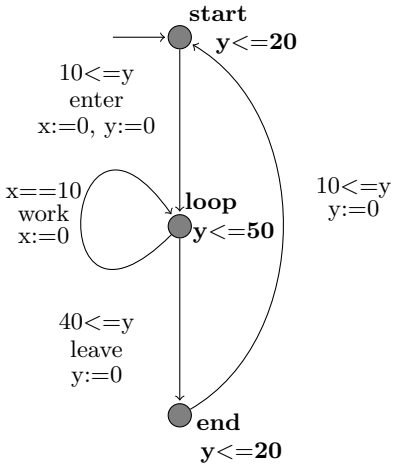
Due to their simplicity, timed safety automata have been used by several tools (e.g., Uppaal) for the simulation and verification of timed automata [1].

Syntax. Assume a finite set of real-valued variables $\mathcal{C}$ ranged over by x, y, ... standing for clocks, and a finite alphabet Σ ranged over by a, b, ... standing for actions. A clock constraint is a conjunctive formula of constraints of the form $x \sim m$ or $x - y \sim m$, for $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$, and $m \in \mathbb{N}$. The set of clock constraints, ranged over by g, is denoted by $\mathcal{B}(\mathcal{C})$.

Definition 1. *A* **timed safety automaton** $\mathcal{A}$ *is a tuple* $\langle N, n_0, E, I\rangle$*, where*

- N *is a finite set of nodes;*
- n_0 *is the initial node;*
- $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$ *is the set of edges;*
- $I : N \rightarrow \mathcal{B}(\mathcal{C})$ *assigns invariants to nodes.*

$n \xrightarrow{g,a,r} n'$ *is a shorthand notation for* $\langle n, g, a, r, n'\rangle \in E$*. Node invariants are restricted to constraints of the form:* $x \leq m$ *or* $x < m$ *where* $m \in \mathbb{N}$*.*



In other words, a timed safety automata is a graph having a finite set of nodes and a finite set of labelled edges (representing transitions), using real-timed variables (representing the clocks of the system). The clocks are initialised with zero when the system starts, and then increased synchronously with the same rate. The behaviour of the automaton is restricted by using clock constraints, i.e. guards on edges, and local timing constraints called *node invariants* (e.g., see Figure 3). An automaton is allowed to stay in a node as long as the timing conditions of that node are satisfied. A transition can be taken when the edge guards are satisfied by clocks values. When a transition is taken, clocks may be reset to zero.

Fig. 3. Timed Safety Automata

Networks of Timed Automata. A network of timed automata is the parallel composition $\mathcal{A}_1 \mid \ldots \mid \mathcal{A}_n$ of a set of timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ combined into a single system using the CCS-like parallel composition operator and with all internal actions hidden. Synchronous communication inside the network is by handshake synchronisation of input and output actions. In this case, the action alphabet Σ consists of $a?$ symbols (for input actions), $a!$ symbols (for output actions), and τ symbols (for internal actions). A detailed example is found in [15].

A network can perform delay transitions (delay for some time), and action transitions (follow an enabled edge). An action transition is enabled if the clock assignment also satisfies all integer guards on the corresponding edges. In synchronisation transitions, the resets on the edge with an output-label are performed before the resets on the edge with an input-label. To model urgent synchronisation transitions that should be taken as soon as they are enabled (the system may not delay), a notion of urgent channels is used. 1-to-many synchronisations are possible using broadcast channels: an edge with synchronisation label $a!$ emits a broadcast and any enabled edge with synchronisation label $a?$ synchronises with the emitting automata.

Let u, v, ... denote clock assignments mapping $\mathcal{C}$ to non-negative reals $\mathbb{R}_+$. Even though u and v and N are overloaded (we keep the initial notations in both formalisms), they are understood according to their context. $g \models u$ means that the clock values u satisfy the guard g. For $d \in \mathbb{R}_+$, the clock assignment mapping all $x \in \mathcal{C}$ to $u(x) + d$ is denoted by $u + d$. Also, for $r \subseteq \mathcal{C}$, the clock assignment mapping all clocks of r to 0 and agreeing with u for the other clocks in $\mathcal{C} \backslash r$ is denoted by $[r \mapsto 0]u$. Let n_i stand for the ith element of a node vector n, and $n[n'_i/n_i]$ for the vector n with n_i being substituted with n'_i.

A network state is a pair $\langle n, u\rangle$, where n denotes a vector of current nodes of the network (one for each automaton), and u is a clock assignment storing the current values of all network clocks and integer variables.

Definition 2. *The operational semantics of a timed automaton is a transition system where states are pairs $\langle n, u\rangle$ and transitions are defined by the rules:*

- $\langle n, u\rangle \xrightarrow{d} \langle n, u + d\rangle$ *if* $u \in I(n)$ *and* $(u + d) \in I(n)$, *where* $I(n) = \bigwedge I(n_i)$;
- $\langle n, u\rangle \xrightarrow{\tau} \langle n[n'_i/n_i], u'\rangle$ *if* $n_i \xrightarrow{g,\tau,r} n'_i$, $g \models u$, $u' = [r \mapsto 0]u$ *and* $u' \in I(n[n'_i/n_i])$;
- $\langle n, u\rangle \xrightarrow{\tau} \langle n[n'_i/n_i][n'_j/n_j], u'\rangle$ *if there exist* $i \neq j$ *such that*
  1. $n_i \xrightarrow{g_i,a?,r_i} n'_i$, $n_j \xrightarrow{g_j,a!,r_j} n'_j$, $g_i \wedge g_j \models u$,
  2. $u' = [r_i \mapsto 0]([r_j \mapsto 0]u)$ *and* $u' \in I(n[n'_i/n_i][n'_j/n_j])$.

4 Relating rTiMo to Timed Safety Automata

In order to use well-known tools such as Uppaal for the verification of distributed networks with migration and communication, we establish a relationship between rTiMo and timed safety automata.

Building a timed safety automaton for each located process: Given a component $l[[P]]$ of an rTiMo network, we associate to it a timed safety automaton $\mathcal{A} = \langle N, n_0, E, I\rangle$ with a local clock x, where $n_0 = l_0$, $N = \{l_0\}$, $E = \emptyset$, $I = \emptyset$. The nodes of the associated automata are labelled using the current location of the located process P (l in this case), and an index such that the nodes are uniquely labelled in this automaton (we start with the index 0, and increment it when necessary). Thus, l_0 means that we model a located process running at location l in rTiMo. The components N, E and I are updated depending on the structure of process P:

- for $P = a^{\Delta t}!\langle v\rangle$ then P_1 else P'_1 we have
  - $N = N \cup \{l_{i+1}, l_{i+2}\}$;
    - If P is running at location l, and N contains some indexed nodes l, namely $l_0, \ldots, l_i$ then add l_{i+1} and l_{i+2} to N. The two nodes indicate the two executions of the located process P, leading to either P_1 or P'_1.
  - $E = E \cup \{n, x < t, a!, x = 0, l_{i+1}\} \cup \{n, x == t, \tau, x = 0, l_{i+2}\}$;
    - If process P is running at location l, and $i > 0$ it means that the automaton already contains some edges, and a process P was launched from the then or else branch of a process P'. Since the translation is made depending on the structure of the processes, it means that the action leading to P is already modelled in the automaton. If $P' = b^{\Delta t'}!\langle w\rangle$ then P else P'' or $P' = b^{\Delta t'}!\langle w\rangle$ then P'' else P or $P' = b^{\Delta t'}?(x)$ then P else P'' or $P' = b^{\Delta t'}?(x)$ then P'' else P or $P' = go^{\Delta t'}l$ then P, then the action of P' is modelled by an edge with the last component l_k, and thus $n = l_k$.

  * Otherwise, $n = l_0$.

  The edge $\{n, x < t, a!, x = 0, l'_{i+1}\}$ encodes the then branch leading to process P_1, while the edge $\{n, x == t, \tau, x = 0, l_{i+2}\}$ encodes the else branch leading to process P'_1. Channel a is an urgent channel (communication takes place as soon as possible).
- $I(n) = \{x <= t\}$.
  * The process should communicate before a maximum of t units of time have elapsed.

– for $P = a^{\Delta t}?(y)$ then P_1 else P'_1 we have
- $N = N \cup \{l_{i+1}, l_{i+2}\}$;
  * If P is running at location l, and N contains some indexed nodes l (namely $l_0, \ldots, l_i$), then add l_{i+1} and l_{i+2} to N. The two nodes indicate the two executions of the located process P, leading either to P_1 or P'_1.
- $E = E \cup \{n, x < t, a?, \{x = 0, y = v\}, l_{i+1}\} \cup \{n, x == t, \tau, x = 0, l_{i+2}\}$;
  * If process P is running at location l and $i > 0$, using a similar argument as for the output action, it holds that $n = l_k$.
  * Otherwise, $n = l_0$.

  The edge $\{n, x < t, a?, \{x = 0, y = v\}, l'_{i+1}\}$ encodes the then branch leading to process P_1, while the edge $\{n, x == t, \tau, x = 0, l_{i+2}\}$ encodes the else branch leading to process P'_1. In order to use an assignment $y = v$ on the edge with $a?$, we impose the condition that channel a can be used at most once for output actions in the translated rTiMo network. This requirement reflects somehow the global asynchronicity of distributed systems (as it is described formally in process calculi).
- $I(n) = \{x <= t\}$.
  * The process should communicate before a maximum of t units of time have elapsed.

– for $P = go^{\Delta t} l'$ then P' we have
- $N = N \cup \{l'_j\}$;
  * If N contains indexed nodes l' (namely $l'_0, \ldots, l'_{j-1}$), then add l'_j to N.
  * Otherwise, add l'_0 to N.

  The new node indicates the execution of process P leading to P'.
- $E = E \cup \{n, x == t, \tau, x = 0, l'_j\}$;
  * If process P is running at location l and $i > 0$, using a similar argument as for the communication actions, it holds that $n = l_k$.
  * Otherwise, $n = l_0$.
- $I(n) = \{x <= t\}$.
  * The process should leave location n before a maximum of t units of time have elapsed.

– for $P = 0$ we have
- N, E and I remain unchanged, and the construction of $\mathcal{A}$ stops.

– for $P = id(v)$ we have
- N remains the same;

- $E = E \cup \{n, x == 0, \tau, \{x = 0, varid = v\}, l_0\}$;
  - If process P is running at location l and $i > 0$, using a similar argument as for the communication actions, it holds that $n = l_k$.
  - Otherwise, $n = l_0$.
- $I(n) = \{x <= 0\}$.
  - The process should leave location n in maximum of 0 units of time.

– for $P = P_1 \mid \ldots \mid P_k, k > 1$, and P_j does not contain operator $\mid$ at top level, then
- $N = N \cup \{l_{i+1}\}$;
  - If P is running at location l, and N contains some indexed nodes l (namely $l_0, \ldots, l_i$), then add l_{i+1} to N.
- $E = E \cup \{n, , a!, \{x = 0\}, l_{i+1}\}$;
  - If process P is running at location l and $i > 0$, using a similar argument as for the communication actions, it holds that $n = l_k$. We use a new channel labelled a as a broadcast channel, in order to start at the same time all the parallel processes from P.
  - Otherwise, $n = l_0$.

  The new edge leads to process P_1. For each of the other processes P_j, $j > 1$, a new automaton $\mathcal{A}_j = \langle N_j, n_{j0}, E_j, I_j \rangle$ is constructed, where:
  - $n_{j0} = l_0$; $N_j = \{l_0, l_1\}$; $E_j = \{l_0, , a?, \{x = 0\}, l_1\}$; $I_j(l_0) = \emptyset$.

  This automaton is constructed then recursively using the definition of P_j.
- $I(n) = \{x <= 0\}$.
  - The process has to communicate in maximum of 0 units of time.

Building a timed automaton for each located process leads to the next result about the equivalence between an rTiMo network N and its corresponding timed safety automaton $\mathcal{A}_N$ in state $\langle n_N, u_N \rangle$ (i.e., $(\mathcal{A}_N, \langle n_N, u_N \rangle)$. Their transition systems differ not only in transitions, but also in states; thus, we adapt the notion of bisimilarity:

Definition 3. *A symmetric relation $\sim$ over* TiMo *networks and the timed safety automata, is a bisimulation if whenever $(N, (\mathcal{A}_N, \langle n_N, u_N \rangle)) \in \sim$:*

– *if $N \xrightarrow{\lambda} N'$, then $\langle n_N, u_N \rangle \xrightarrow{\tau} \langle n_{N'}, u_{N'} \rangle$ and $(N', (\mathcal{A}_{N'}, \langle n_{N'}, u_{N'} \rangle)) \in \sim$ for some N'.*
– *if $N \overset{t}{\rightsquigarrow} N'$, then $\langle n_N, u_N \rangle \xrightarrow{d} \langle n_{N'}, u_{N'} \rangle$ and $(N', (\mathcal{A}_{N'}, \langle n_{N'}, u_{N'} \rangle)) \in \sim$ for some N', where $u_{N'} = u_N + d$.*

Having defined bisimulation, we can state our main theorem as follows.

Theorem 1. *Given an* rTiMo *network N with channels appearing only once in output actions, there exists a timed safety automaton $\mathcal{A}_N$ with a bisimilar behaviour. Formally, $N \sim \mathcal{A}_N$.*

Proof (Sketch). The construction of the timed safety automaton simulating a given rTiMo network is presented above. Due to the limitations of Uppaal , we imposed the requirement of using at most once an output actions in order to allow the assignment $y = v$ on edges with input labels (as used in the building of the automaton).

A bisimilar behaviour is given by:

- at the start of execution, all clock in rTiMo and their corresponding timed automata are set to 0;
- the consumption of a *go* action in a node l_i is matched by an τ edge obtained by translation;
- a communication rule is matched by a synchronisation between the edges obtained by translations;
- the passage of time is similar in both formalisms: in rTiMo the global clock is used to decrement by d all timers in the network when no action is possible, while in the timed automata all local clocks are decremented synchronously with the same value d when no edge can be taken.

Thus, the size of a timed safety automata $\mathcal{A}_N$ is polynomial with respect to the size of a TiMo network N, and the state spaces have the same number of states.

Reachability Analysis. One of the most useful question to ask about a timed automaton is the reachability of a given set of final states. Such final states may be used to characterise safety properties of a system.

Definition 4. *We write* $\langle n, u\rangle \rightarrow \langle n', u'\rangle$ *whenever* $\langle n, u\rangle \xrightarrow{\sigma} \langle n', u'\rangle$ *for* $\sigma \in \Sigma \cup \mathbb{R}_+$. *For an automaton with initial state* $\langle n_0, u_0\rangle$, $\langle n, u\rangle$ *is reachable if and only if* $\langle n_0, u_0\rangle \rightarrow^* \langle n, u\rangle$. *More generally, given a constraint* $\phi \in \mathcal{B}(\mathcal{C})$ *if* $\langle n, u\rangle$ *is reachable for some* u *satisfying* ϕ *then a state* $\langle n, \phi\rangle$ *is reachable.*

Invariant properties can be specified using clock constraints in combination with local properties on nodes. The reachability problem is decidable [5].

The reachability problem can be also defined for rTiMo networks.

Definition 5. *We write* $N \rightarrow N'$ *if* $N \xrightarrow{\lambda} N'$ *or* $N \overset{t'}{\rightsquigarrow} N'$ *for actions* λ *of the form* $l' \triangleright l$, $\{v/u\}@l$, $id@l$, $go^{\Delta 0}@l$, $a?^{\Delta 0}@l$ *or* $a!^{\Delta 0}@l$. *Starting from an* rTiMo *network* N_0, *a configuration* N_1 *is reachable if and only if* $N_0 \rightarrow^* N_1$.

The following result is a consequence of Theorem 1.

Corollary 1. *For an* rTiMo *network with channels appearing only once in output actions, the reachability problem is decidable.*

Bisimulation. Two timed automata are defined to be timed bisimilar in [5] if and only if they perform the same action transitions and reach bisimilar states.

Definition 6. *A symmetric relation* $\mathcal{R}$ *over the timed automata and the alphabet* $\Sigma \cup \mathbb{R}_+$, *is a bisimulation if:*

- *for all* $(s_1, s_2) \in \mathcal{R}$, *if* $s_1 \xrightarrow{\sigma} s_1'$ *for* $\sigma \in \Sigma \cup \mathbb{R}_+$ *and* s_1', *then* $s_2 \xrightarrow{\sigma} s_2'$ *and* $(s_1', s_2') \in \mathcal{R}$ *for some* s_2'.

Proposition 2. *[6] Timed bisimulation is decidable.*

In a similar way to our previous approach in [2], we define the bisimulation over configurations of rTiMo networks.

Definition 7. *A symmetric relation $\mathcal{R}$ over the* rTiMo *networks and the set Act of actions, is a bisimulation if:*

- *for all $(N_1, N_2) \in \mathcal{R}$, if $N_1 \xrightarrow{\lambda} N_1'$ for $\lambda \in Act$ and N_1', then $N2 \xrightarrow{\lambda} N_2'$ and $(N_1', N_2') \in \mathcal{R}$ for some N_2'.*
- *for all $(N_1, N_2) \in \mathcal{R}$, if $N_1 \overset{t}{\rightsquigarrow} N_1'$ for $t \in \mathbb{N}$ and N_1', then $N_2 \overset{t}{\rightsquigarrow} N_2'$ and $(N_1', N_2') \in \mathcal{R}$ for some N_2'.*

The following result is a consequence of Theorem 1.

Corollary 2. *For two* rTiMo *networks with channels appearing only once in output actions, timed bisimulation is decidable.*

5 Verifying Properties of rTiMo by Using Uppaal

By virtue of the results presented in the previous section, we can verify real-time systems corresponding to a subclass of rTiMo networks by using Uppaal. In general, modelling and verification of real-time systems in Uppaal were presented in [16]. Uppaal can be used to check temporal properties of networks of timed safety automata, properties expressed in Computation Tree Logic (CTL). If ϕ and ψ are boolean expressions over predicates on nodes, integer variables and clock constraints, then the formulas have the following forms:

$\mathsf{A}\,[\,]\,\phi$ - Invariantly ϕ; $\mathsf{A}\,\langle\,\rangle\,\phi$ - Always Eventually ϕ;
$\mathsf{E}\,[\,]\,\phi$ - Potentially Always ϕ; $\mathsf{E}\,\langle\,\rangle\,\phi$ - Possibly ϕ;
$\phi \rightsquigarrow \psi$ - ϕ always leads to ψ. This is a shorthand for $\mathsf{A}\,[\,]\,(\phi \Rightarrow \mathsf{A}\,\langle\,\rangle\,\psi)$

The properties most commonly used in verification of timed systems are $\mathsf{E}\,\langle\,\rangle\,\phi$ and $\mathsf{A}\,[\,]\,\phi$. They represent safety properties (a specified error can not occur).

The properties $\mathsf{A}\,\langle\,\rangle\,\phi$, $\mathsf{E}\,[\,]\,\phi$ and $\phi \rightsquigarrow \psi$ represent unbounded liveness properties (used to express and check global progress), and are not commonly used in Uppaal case studies. Bounded properties are important for timed systems.

Example 2. Using both types of properties, we performed some verifications in Uppaal for the system presented in Example 1.

- E[ ]clientcash <= 0
  Checks if potentially always on some path *clientcash* <= 0 is not satisfied.
- A <> till.bank1 imply till.x >= 1
  Checks if the *till* automaton is in the *bank* node, then the value of the local clock is >= 1.
- E <> (clientcash == 70)&&(agentcash == 170)&&(bankcash == 60)
  Checks if there exists a state containing the configuration of Figure 2.
- A[ ] not deadlock
  Checks that there exists deadlocks. The error is due to state space explosion.

Several other properties of rTiMo systems can be verified by using Uppaal.

Fig. 4. Verification in UPPAAL

6 Conclusion

When modelling distributed systems it is useful to have explicit notions of locations, clocks, explicit migrations and resource management. Various process calculi derived from π-calculus [18] have been proposed to model some of these aspects. Various features were introduced over the basic π-calculus: e.g., explicit locations in distributed π-calculus [14], and explicit migration and timers in timed distributed π-calculus ($tD\pi$) [11]. TIMO [8] is essentially a simplified version of $tD\pi$ designed to allow appropriate software architecture for implementation [7]. TIMO represents an attempt to bridge the gap between the existing (theoretical) process calculi and forthcoming realistic languages for multi-agent systems.

Several proposals for real-time modelling and verification are present in the literature: timed automata [1], timed CSP [20], timed ACP [3], and several timed extensions of CCS [19,21]. In this paper we defined a formalism called rTIMO that uses real-time and explicit timeouts, and so is useful for expressing certain temporal properties of multi-agent systems with migration and time constraints. In order to illustrate in rTIMO the coordination of migrating agents in time and space, we adapt the *TravelShop* example from [9] in which a client attempts to get a ticket to a predefined destination in a short time and/or at a good price. Although the syntax of rTIMO is quite close to that of TIMO , its semantics is different in many aspects: the number of semantic rules, number of clocks, time nature (continuous or discrete), systems evolution.

We established a formal relationship between rTIMO and timed safety automata allowing the use of model checking capabilities provided by UPPAAL to verify several temporal properties of distributed networks with migrating and communicating processes described in rTIMO. The verification performed on the *TravelShop* example also validates the rTIMO semantics.

As future work we intend to use this relationship for improvements in rTIMO (e.g., constraints on integers, lower and upper time bounds) in order to extend the classes of complex systems that can be modelled and analysed.

Acknowledgements. Many thanks to the reviewers for their useful comments. The work was supported by a grant of the Romanian National Authority for Scientific Research, project number PN-II-ID-PCE-2011-3-0919.

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
2. Aman, B., Ciobanu, G., Koutny, M.: Behavioural Equivalences over Migrating Processes with Timers. In: Giese, H., Rosu, G. (eds.) FMOODS/FORTE 2012. LNCS, vol. 7273, pp. 52–66. Springer, Heidelberg (2012)
3. Baeten, J.C.M., Bergstra, J.A.: Real Time Process Algebra. Journal of Formal Aspects of Computing Science 3(2), 142–188 (1991)
4. Baumann, J., Hohl, F., Radouniklis, N., Rothermel, K., Straβer, M.: Communication Concepts for Mobile Agent Systems. In: Rothermel, K., Popescu-Zeletin, R. (eds.) MA 1997. LNCS, vol. 1219, pp. 123–135. Springer, Heidelberg (1997)
5. Bengtsson, J., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
6. Čerāns, K.: Decidability of Bisimulation Equivalences for Parallel Timer Processes. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 302–315. Springer, Heidelberg (1993)
7. Ciobanu, G., Juravle, C.: Flexible Software Architecture and Language for Mobile Agents. Concurrency and Computation: Practice and Experience 24, 559–571 (2012)
8. Ciobanu, G., Koutny, M.: Modelling and Verification of Timed Interaction and Migration. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 215–229. Springer, Heidelberg (2008)
9. Ciobanu, G., Koutny, M.: Timed Mobility in Process Algebra and Petri Nets. Journal of Logic and Algebraic Programming 80, 377–391 (2011)
10. Ciobanu, G., Koutny, M.: Timed Migration and Interaction With Access Permissions. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 293–307. Springer, Heidelberg (2011)
11. Ciobanu, G., Prisacariu, C.: Timers for Distributed Systems. Electronic Notes in Theoretic Computer Science 164(3), 81–99 (2006)
12. Groote, J.F.: Transition System Specifications with Negative Premises. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 332–341. Springer, Heidelberg (1990)
13. Hennessy, M., Regan, T.: A Process Algebra for Timed Systems. Information and Computation 117, 221–239 (1995)
14. Hennessy, M.: A Distributed π-calculus. Cambridge University Press (2007)
15. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic Model Checking for Real-time Systems. Information and Computation 111, 192–224 (1994)
16. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing Real-Time Systems Using UPPAAL. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 77–117. Springer, Heidelberg (2008)
17. Larsen, K.G., Petterson, P., Yi, W.: Uppaal in a Nutshell. International Journal on Software Tools for Technology Transfer 1(2), 134–152 (1997)
18. Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press (1999)
19. Moller, F., Tofts, C.: A Temporal Calculus of Communicating Systems. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 401–415. Springer, Heidelberg (1990)
20. Reed, G.M., Roscoe, A.W.: A Timed Model for Communicating Sequential Processes. Theoretical Computer Science 58(1-3), 249–261 (1988)
21. Yi, W., Pettersson, P., Daniels, M.: Automatic Verification of Real-time Communicating Systems by Constraint-solving. In: International Conference on Formal Description Techniques, pp. 223–238 (1994)